
APACHE
Spark
TM

Agenda

- What is Apache Spark?
- Fundamentals
- Why Spark?
- Spark Components
- Spark Concepts
- Spark Lifecycle
- Spark Stack
- Spark Input Data
- Spark Apps & Distributors
- Spark Universe
- PySpark
- Spark Competitors

What is Apache Spark?

"Apache Spark is a fast and general engine for large-scale data processing"
https://spark.apache.org/ (https://spark.apache.org/)

- Open source
- Flexible in-memory framework for batch & (near) real-time processing on clusters
- Parallel operations and fault-tolerant

Spark Specs

- Developed: 2009 UC Berkeley's AMPLab by Matei Zaharia
- Open source: 2010 under BSD license
- Donated to Apache: 2013
- Initial release: 2014
- Current stable release: v2.2.0 (July 11, 2017)
- Written in: Scala (77%), Java (10%), Python (8%), R (4%), Other (1%)
- License: Apache License 2.0
- Website: https://spark.apache.org/ (https://spark.apache.org/)
- Repository: https://github.com/apache/spark (https://github.com/apache/spark)

Fundamentals

Apache Hadoop

"Open-source software for reliable, scalable, distributed computing"
http://hadoop.apache.org/ (http://hadoop.apache.org/)

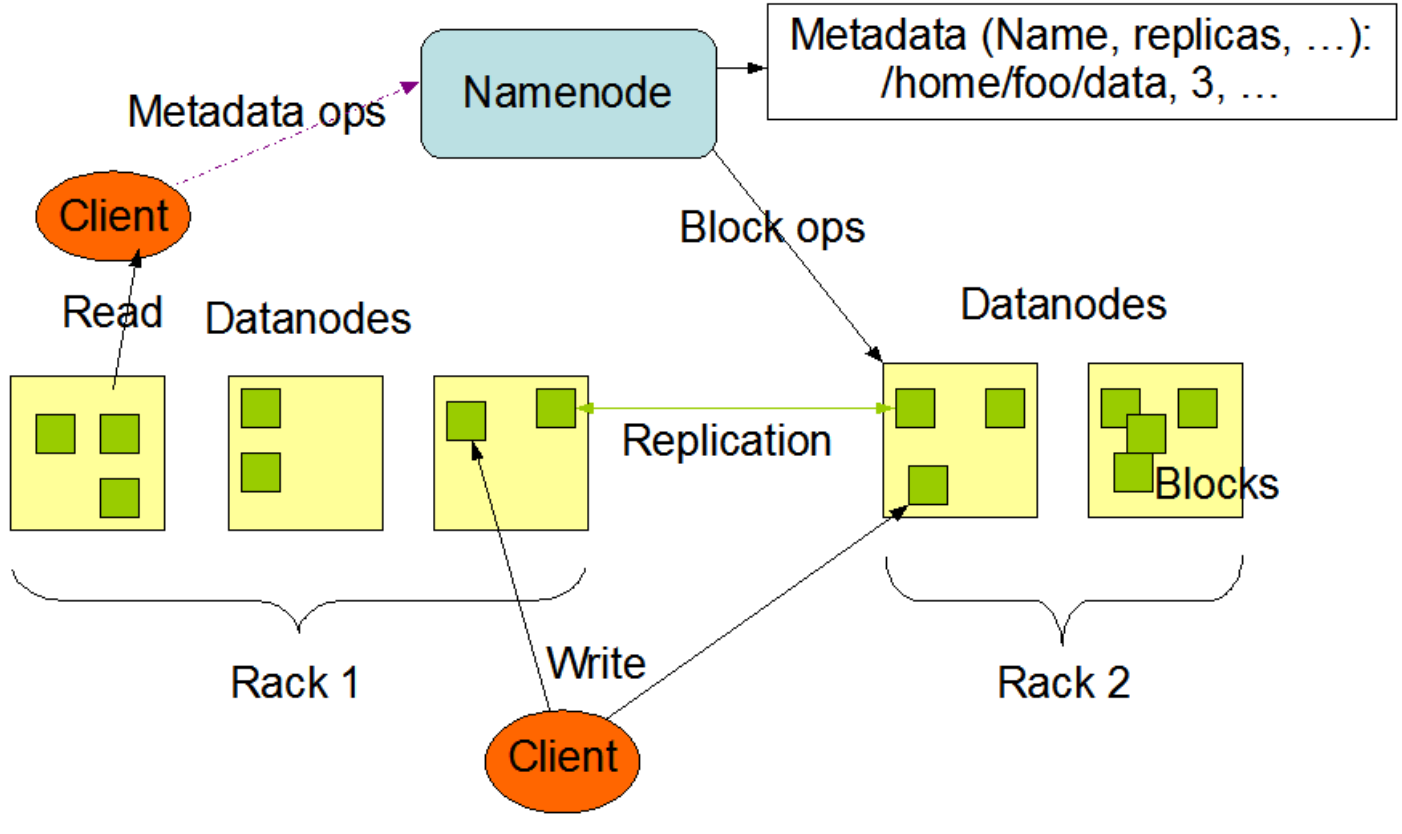
	Hadoop 1.x	Hadoop 2.x	()
			Data Processing
Resource Management & Data Processing			Resource Management
File System			File System

Hadoop 3.0.0-beta1 published on October 3, 2017.

Apache HDFS

- Distributed file system
- Any kind of data
- **Inexpensive:** runs on commodity hardware
- **Reliable:** highly fault-tolerant
- **Scalable:** high throughput for large datasets

HDFS Architecture

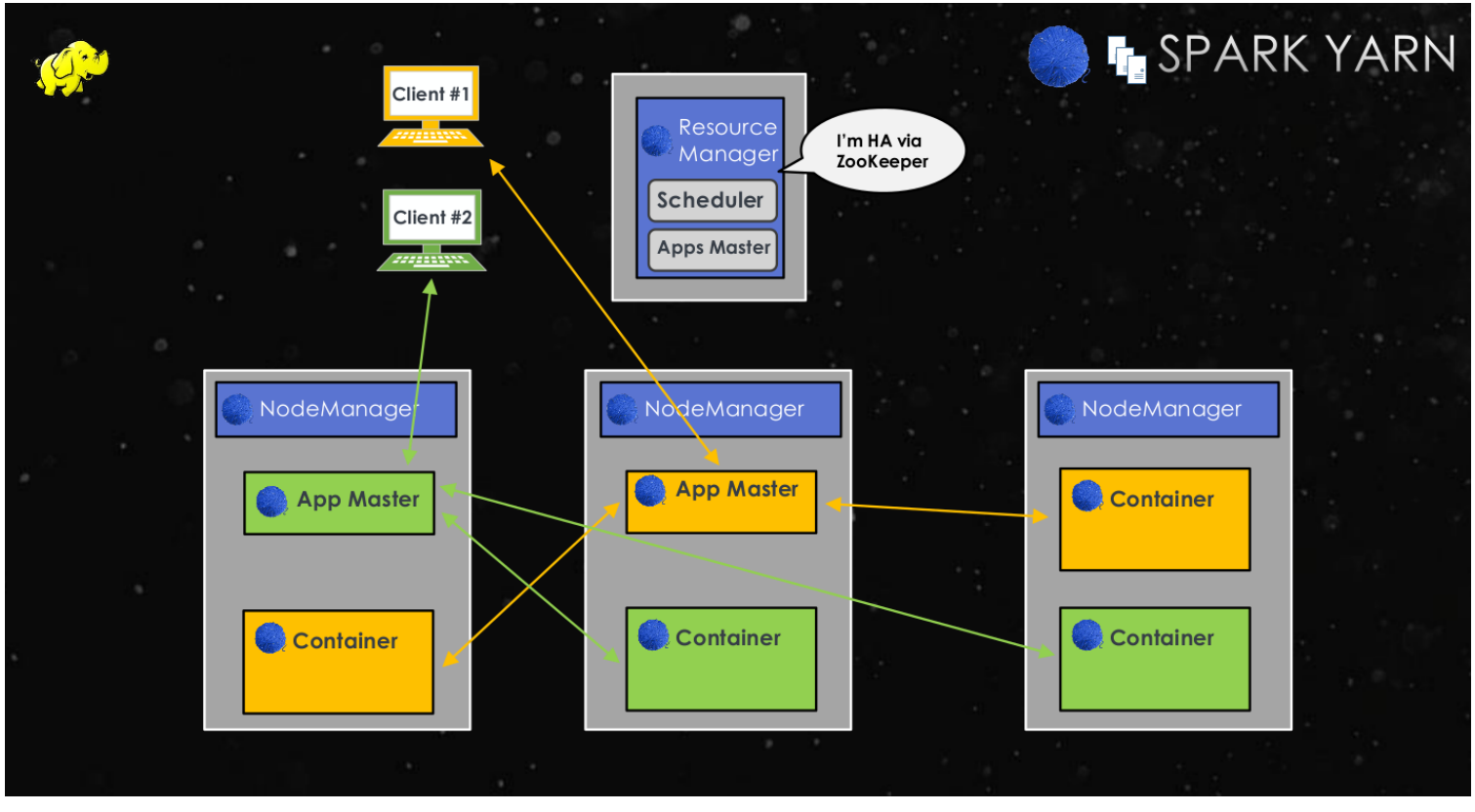


https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
(https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html)

YARN (Yet Another Resource Nagivator)

"Framework for job scheduling and cluster resource management"
https://hadoop.apache.org/ (https://hadoop.apache.org/)

- Integrated into Hadoop 2.0
- Allows multiple applications to run on the same platform
- Components:
  - ResourceManager
  - NodeManager
  - ApplicationMaster


SPARK YARN
Client #1
Client #2
Resource Manager
I'm HA via ZooKeeper
Scheduler
Apps Master
NodeManager
App Master
Container
NodeManager
App Master
Container
NodeManager
Container
Container

MapReduce

"YARN-based system for parallel processing of large datasets"
https://hadoop.apache.org/ (https://hadoop.apache.org/)

- Written in **Java**
- MapReduce job:
  1. Read input data and split input dataset into independent chunks and distribute them
  2. Map task
  3. Shuffle: sort outputs of the maps (=input of reduce tasks)
  4. Reduce task
  5. Write results to disk
- User: specify input/output location and map & reduce functions
- **Batch processing**
- Build around an **acyclic data flow model**: one-pass computations

MapReduce vs Spark in the Apache Universe

	MapReduce for Batch Processing	Spark for In-memory Processing
Data Processing		
Resource Management		
File System		

Why Spark?

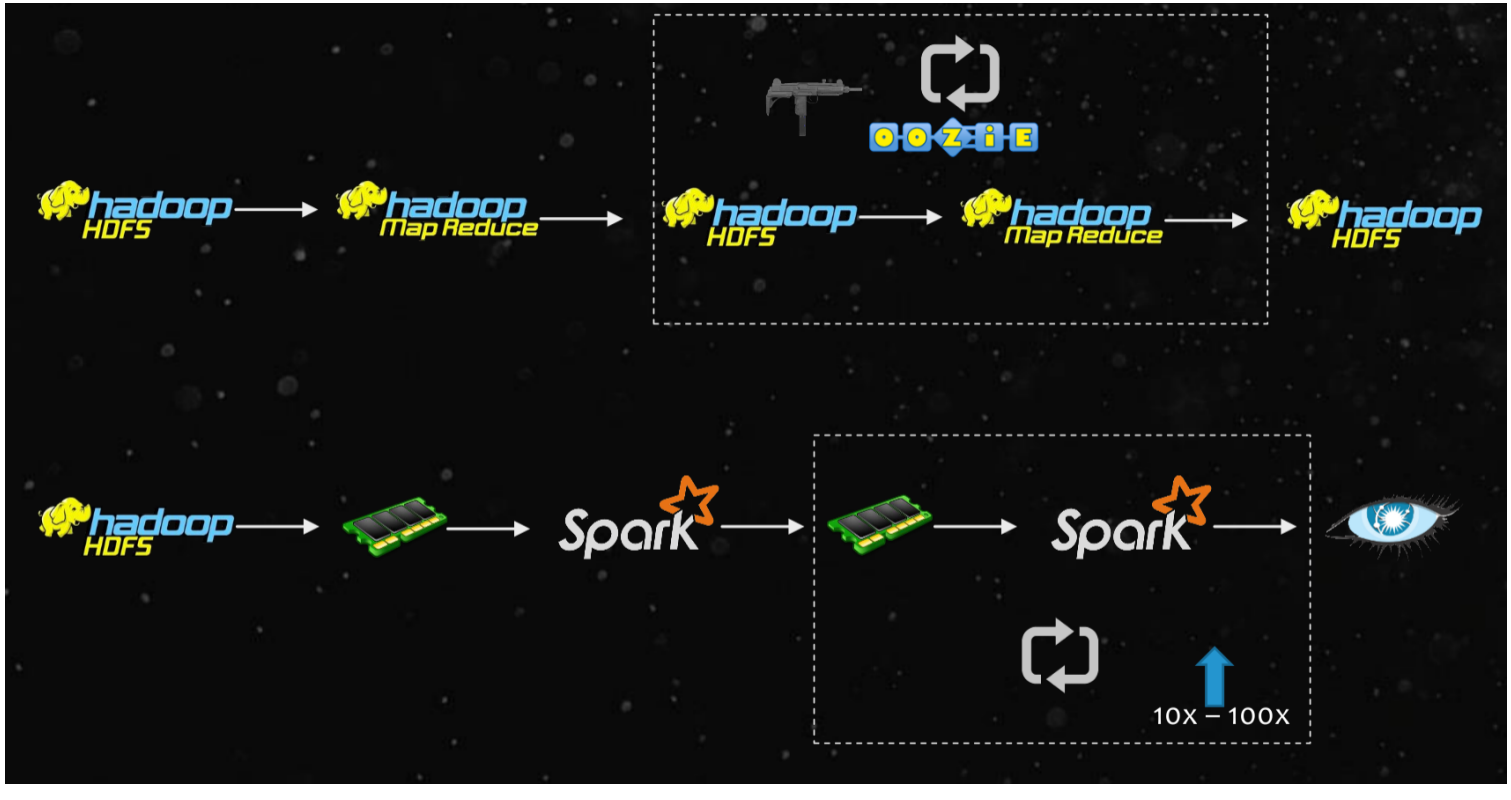
- Speed
- Ease of use
- Generality
- Runs everywhere

Speed

"Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk" https://spark.apache.org/ (https://spark.apache.org/)

- Next Gen Shuffle
  - 100 TB Daytona Sort Competition 2014
  - Sorting on disk (HDFS)
  - 3x faster using 10x fewer machines than Hadoop MapReduce

Iterative Process in MapReduce and Spark



Ease of Use

Word count in Spark's Python API:

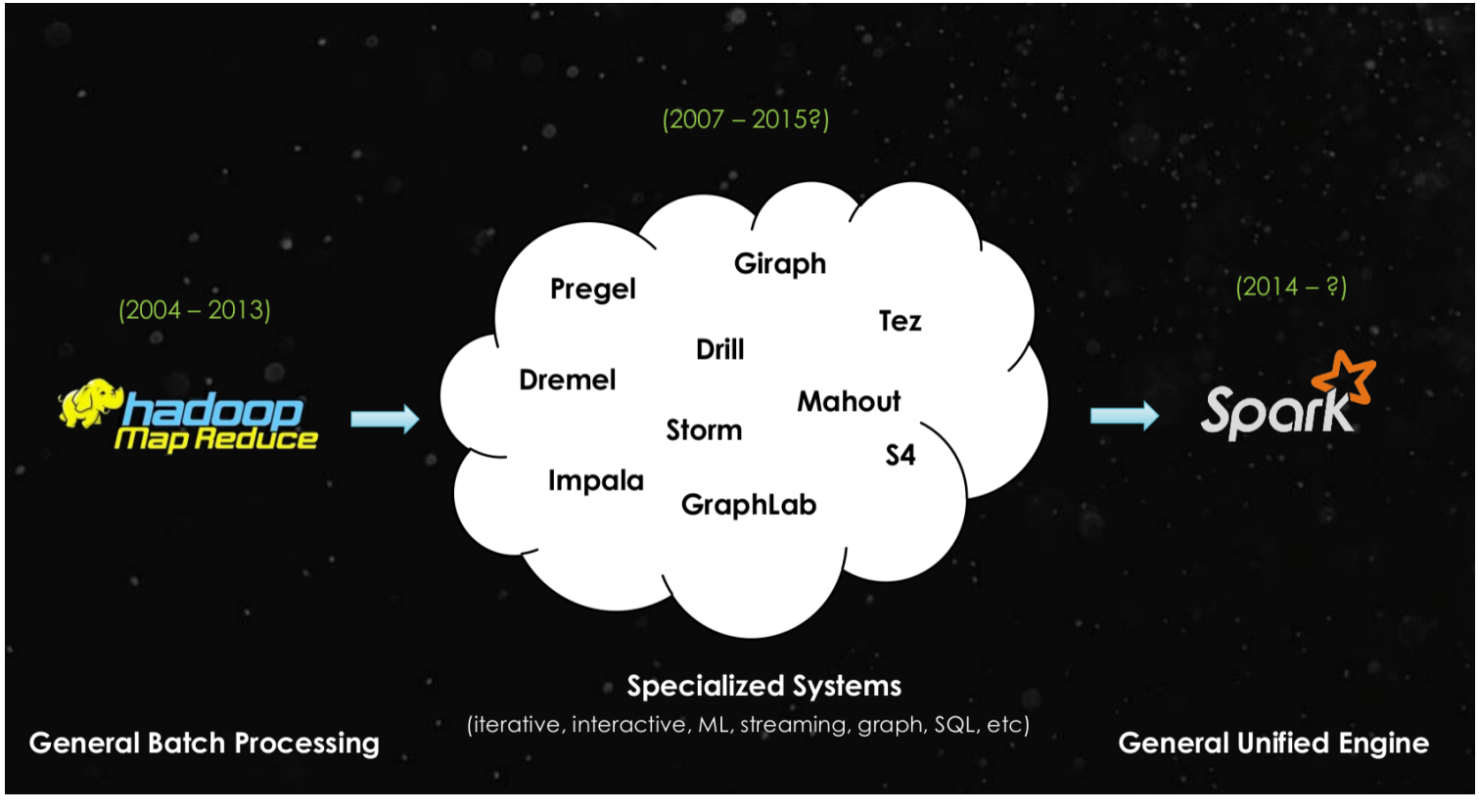
```
text_file = spark.textFile("hdfs://...")

text_file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a+b)
```

https://spark.apache.org/ (https://spark.apache.org/)

Generality

"Combine SQL, streaming, and complex analytics" https://spark.apache.org/ (https://spark.apache.org/)



Runs Everywhere

- Local
- Standalone
- Hadoop Yarn
- Apache Mesos

https://spark.apache.org/ (https://spark.apache.org/)

Spark Components

Driver Program (SparkContext object)

- Process running the user code (creates SparkContext)
  - Converting a program to tasks (Direct Acyclic Graph (DAG))
    - Logical sequence of operations
    - *"The magic behind Spark"*
  - Scheduling tasks on executors

Spark Components

Executors & Worker Nodes

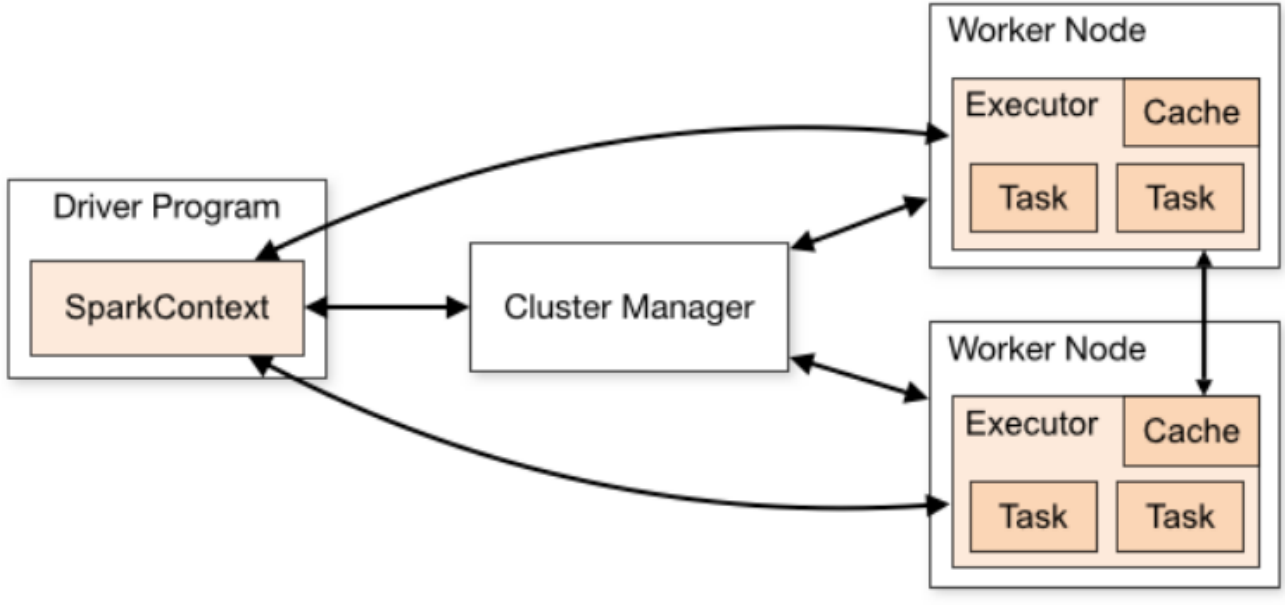
- One worker node per machine but multiple executors per worker node possible (parallelism)
- Executor tasks:
  - Run the individual tasks and return the results to the driver
  - Provide in-memory storage
- Local mode: Spark driver and executor run in the same Java process

Spark Components

Cluster Manager

- Resource allocation
- Communicates with executors and driver
- Partitioning:
  - Static
    - **Local:** local client for prototyping & testing
    - **Standalone:** Spark built-in cluster manager
  - Dynamic
    - **Hadoop Yarn:** Hadoop 2 cluster manager
    - **Apache Mesos:** General cluster manager

Spark Components



Spark Concepts

- RDDs
- Parallel Operations
- Shared Variables

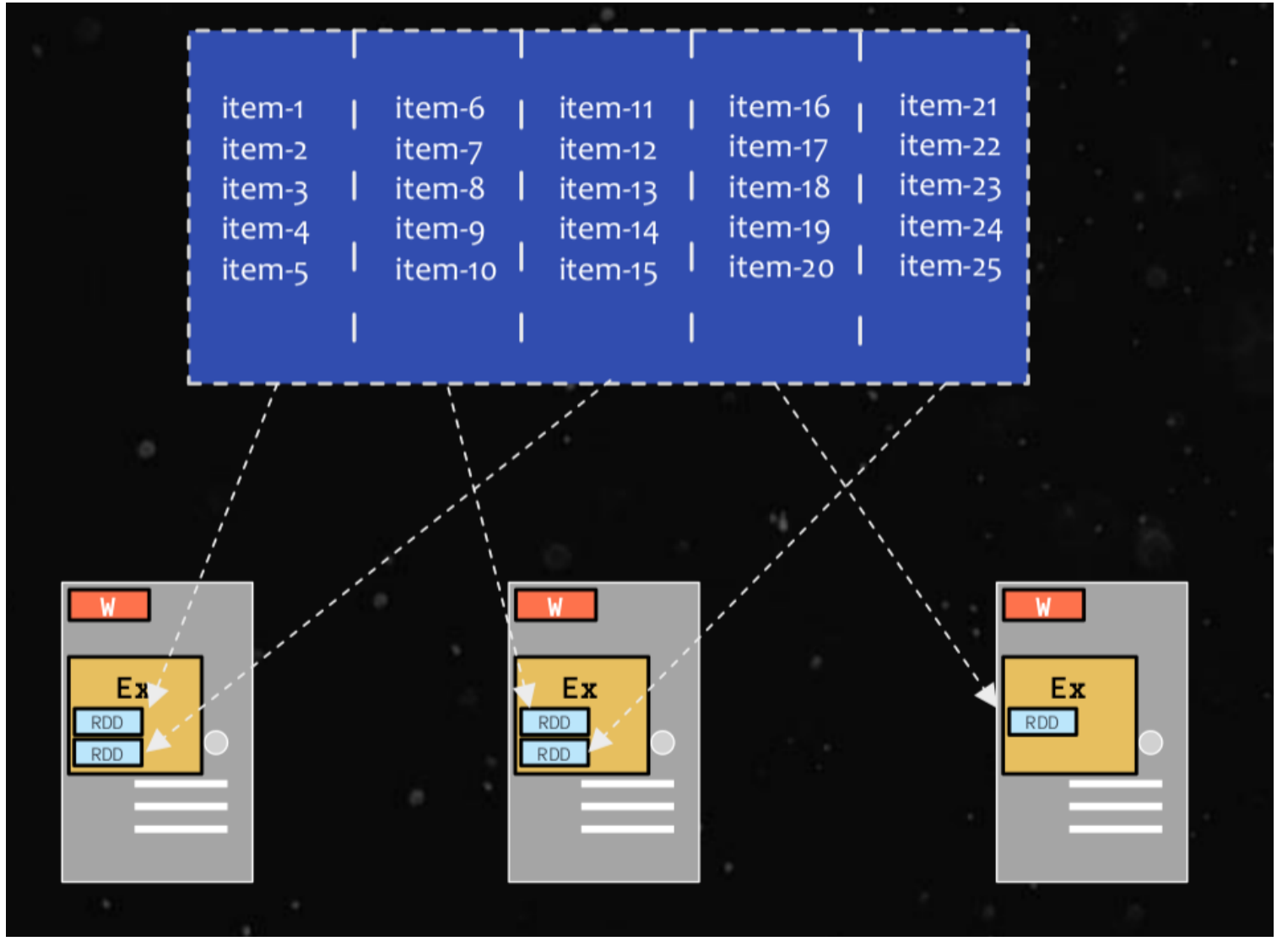
RDDs (Resilient Distributed Datasets)

- Main abstraction in Spark
- **Read-only** collection of objects
- **Partioned** across a set of machines (more partitions = more parallelism)
- **Fault-tolerant:** can be rebuilt if a partition is lost
- **Lazy:** only computed when used

RDDs (Resilient Distributed Datasets)

- Two types of operations:
  - **Transformations:** return a new RDD (e.g. filter)
  - **Actions:** start a computation and return a result (to driver program or write it to storage)
- Can be created:
  - **Parallelizing** a collection: `sc.parallelize(["a", "b", "c"])`
  - **Read data** from disk: e.g. `sc.textFile("/path/f.md")`
  - **Transforming** an existing RDD
  - Change the **persistence** of an existing RDD:
    - cache()
    - save()

RDDs (Resilient Distributed Datasets)



Parallel Operations

- Set of parallel tasks on different nodes
- Copy of each variable used in the function is shipped to each tasks

https://spark.apache.org/docs/latest/rdd-programming-guide.html (https://spark.apache.org/docs/latest/rdd-programming-guide.html)

Shared Variables

"across tasks, or between tasks and the driver program" for parallel operations

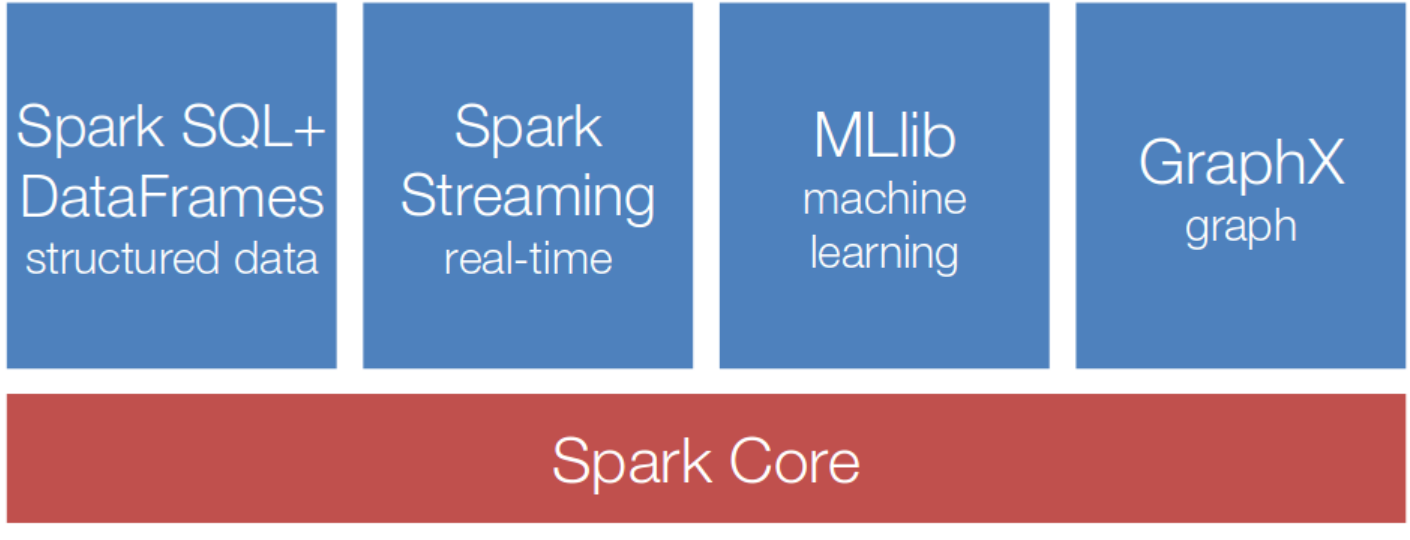
- **Broadcast variables:**
  - Cached in memory on all nodes
  - Only copied once to every worker (e.g. for large read-only lookup tables)
- **Accumulators:**
  - Workers can only "add"
  - Only the driver can read (e.g. for counters)

https://spark.apache.org/docs/latest/rdd-programming-guide.html (https://spark.apache.org/docs/latest/rdd-programming-guide.html)

Spark Lifecycle

1. Create input data source in **RDD** (external or parallelized)
2. (Lazy) **transformations**: define new RDDs
3. [Optional: **cache()** any intermediate RDD for reuse]
4. (Parallel & optimized) **actions**
5. Processed data **RDD** / UI dashboard

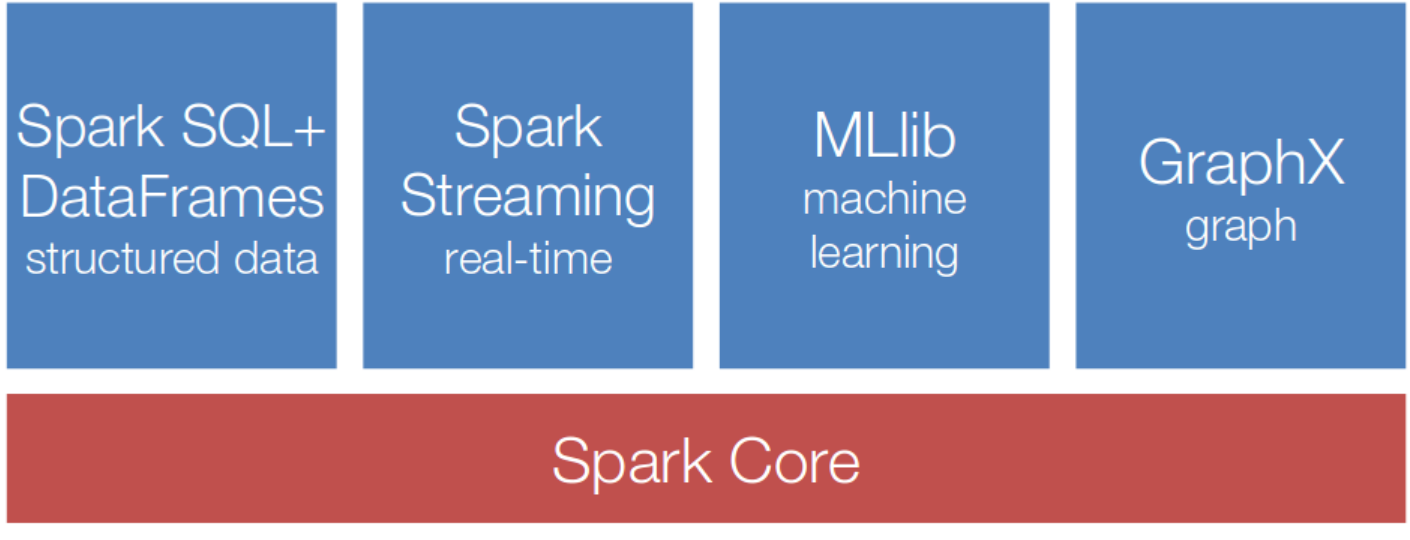
Spark Stack Overview



Spark Core APIs

- Scala
- Java
- Python
- R

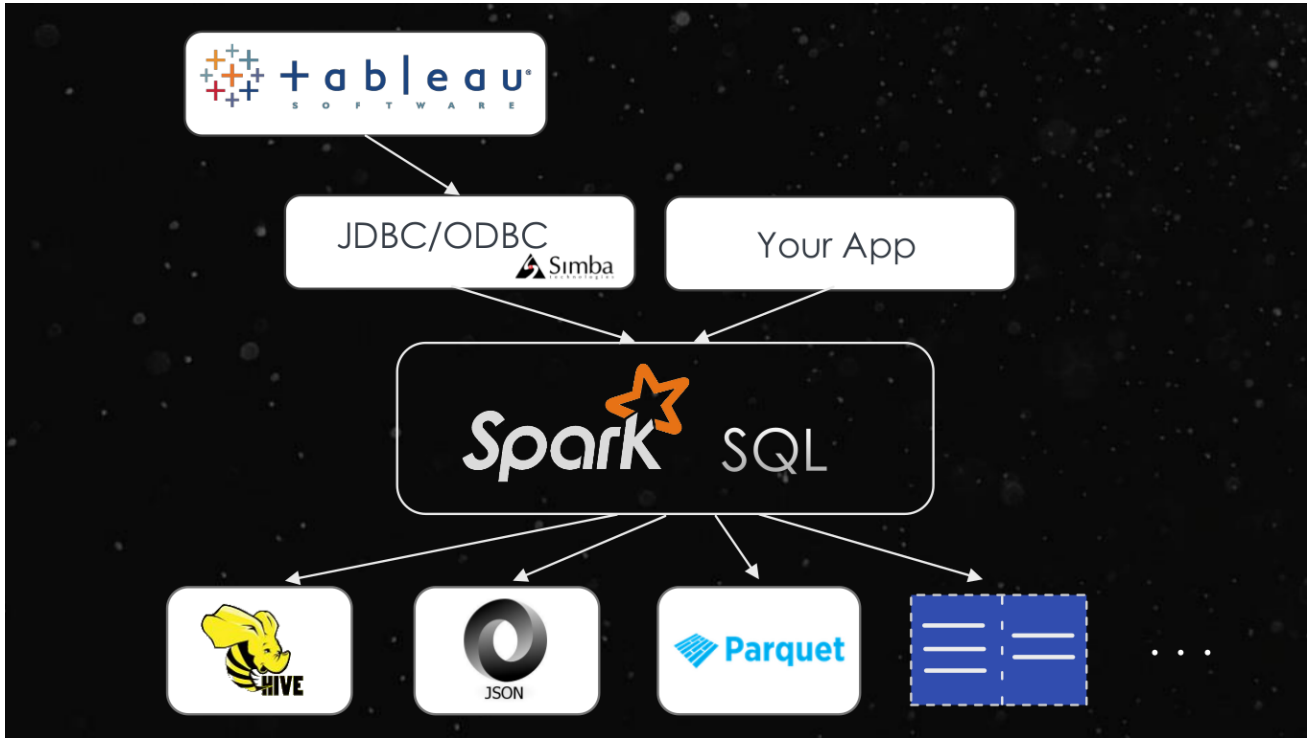
Spark Stack Overview



Spark SQL, DataFrames & Datasets

Spark SQL http://spark.apache.org/sql/ (http://spark.apache.org/sql/)

- Module for working with structured data
  - **Integrated:** SQL queries within Spark programs
  - **Uniform data access:** connect to any data source
  - **Hive integration:** SQL or HiveQL queries
  - **Standard connectivity:** through JDBC or ODBC



Spark SQL, DataFrames & Datasets

RDD Recap

- Low level type-safty (data field types)
- Java / Kyrp serialization
  - Distribute data in network
  - Write data to disk
- Overhead: class structure & values for every record

Spark SQL, DataFrames & Datasets

DataFrame

- **Higher level abstraction** (than RDD) introduced in Spark 1.3
- "Conceptually equivalent to a table in a relational database or a DataFrame in R/Python"
- Schema managed by Spark (**column types**)
- Faster than RDD but **lost type safety**
  - only data (without class structure) send or written
  - optimized relational query plan by Spark's Catelyst optimizer
- DataFrame API in Scala, Java, Python and R
  - Similar to Pandas and R
  - DataFrame is a Dataset of untyped Rows:
    - Scala: `DataFrame = Dataset[Row]`
    - Java: `DataFrame = Dataset<Row>`

http://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes (http://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes)

Spark SQL, DataFrames & Datasets

Dataset

- **"Distributed collection of data"**
- Build on top of RDD (since Spark 1.6)
- **Strongly-typed** to keep track of their schema: `Dataset[T]`
- Combines benefits of RDDs (strong typing & lambda functions) and DataFrames (Spark SQL's optimized execution engine)
- **Tungsten in-memory encoding**: use less memory & fast (de-)serialization
- Manipulated using functional transformations
- Dataset API available in Scala and Java
- Python and R already have many of these benefits by nature (row.columName) but only provide untyped objects

http://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes (http://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes)

Spark SQL, DataFrames & Datasets

```scala
// Read a DataFrame from a JSON file
val df = sqlContext.read.json("people.json")
// Convert the data to a domain object.
case class Person(name: String, age: Long)
val ds: Dataset[Person] = df.as[Person]
```

https://github.com/bmc/rdds-dataframes-datasets-presentation-2016 (https://github.com/bmc/rdds-dataframes-datasets-presentation-2016)

- Spark 2.x unifies concepts of Datasets and DataFrames
  - Untyped (`Dataset[Row]`) and Typed (`Dataset[T]`) Dataset API
- Spark 3.x is expected to remove the RDD API

Spark SQL, DataFrames & Datasets

DataFrames and Datasets: detect syntax and analysis errors at compile time

errors \ API	SQL	DataFrames	Datasets
Syntax	R	C	C
Analysis	R	R	C

R = runtime, C = compile time

http://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes (http://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes)

Spark Streaming

"Build scalable fault-tolerant streaming applications" http://spark.apache.org/streaming/ (http://spark.apache.org/streaming/)

- **Micro-batching:** (near) real-time processing
- **Ease of use:** high-level operators
- **Fault-tolerance:** recovers lost work & operator state
- **Spark integration:** combine streaming with batch & interactive queries
- Unified API for batch (Hadoop MapReduce) and realtime (Apache Storm) processing

Spark Streaming

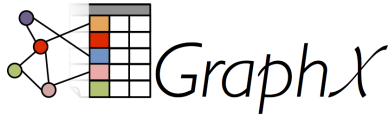
DStream (discretized stream)

- High level abstraction
- Continuous input data stream
- Series of RDDs
- Batch interval: x seconds
  - new RDD is created every x seconds
  - min. 0.5s (90% of the use cases)

MLlib

"Scalable machine learning library" http://spark.apache.org/mllib/ (http://spark.apache.org/mllib/)

- **Easy to use:** Java, Scala, Python, R
- **Performance:** up to 100x faster than MapReduce
- **Easy to deploy:** runs on existing Hadoop clusters & data
- Set of functions to call on DataFrames
- Only contains parallel algorithms



GraphX

"API for graphs & graph-parallel computation"

- Graph abstraction on top of RDDs
- **Flexibility:** works with graphs & collections
- **Speed:** comparable performance with fastest specialized graph processing systems
- **Algorithms:** growing library of graph algorithms

External Projects (extract)

- **SparkR:** R frontend for Spark
- **EclairJS:** use Jupyter notebooks for Spark
- **BlinkDB:** approximate query engine
- **Tachyon:** memory speed virtual distributed storage system

http://spark.apache.org/third-party-projects.html (http://spark.apache.org/third-party-projects.html)

Spark Input Data

"Anything that has a Hadoop Input Format"

- **File Systems**
  - Local Ext3/4
  - HDFS
  - Amazon S3
  - OpenStack Swift
- **SQL & NoSQL Databases**
  - RDBMS
  - Cassandra
  - MongoDB
  - HBase
  - Neo4j
- **Buffers for Spark Streaming**
  - Flume
  - Kafka

Spark Apps & Distributors

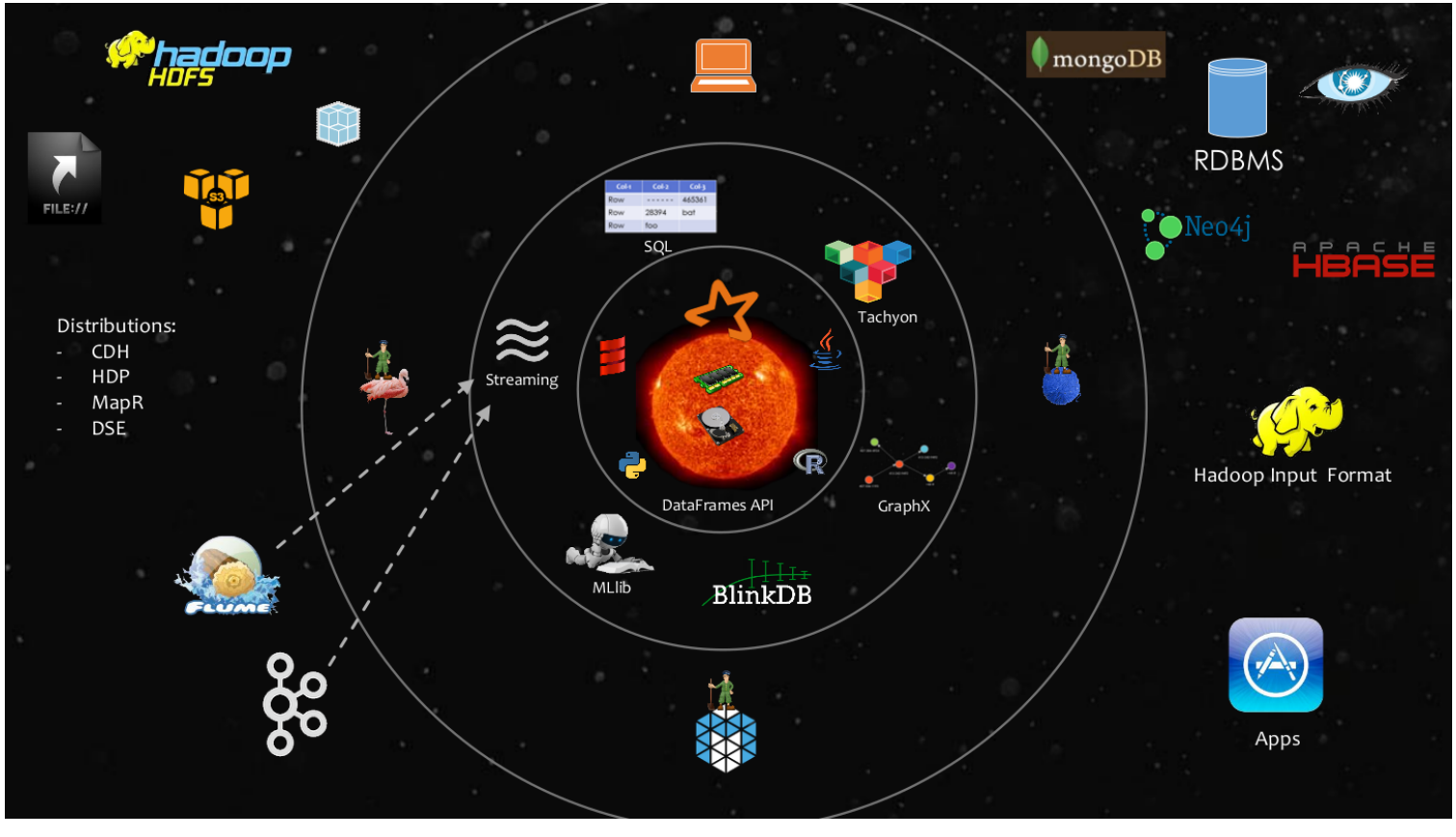
Apps

- Tableau
- elasticsearch
- ...

Distributors

- Datastax
- Cloudera
- Hartonworks
- MapR
- IBM
- SAP
- Oracle
- ...

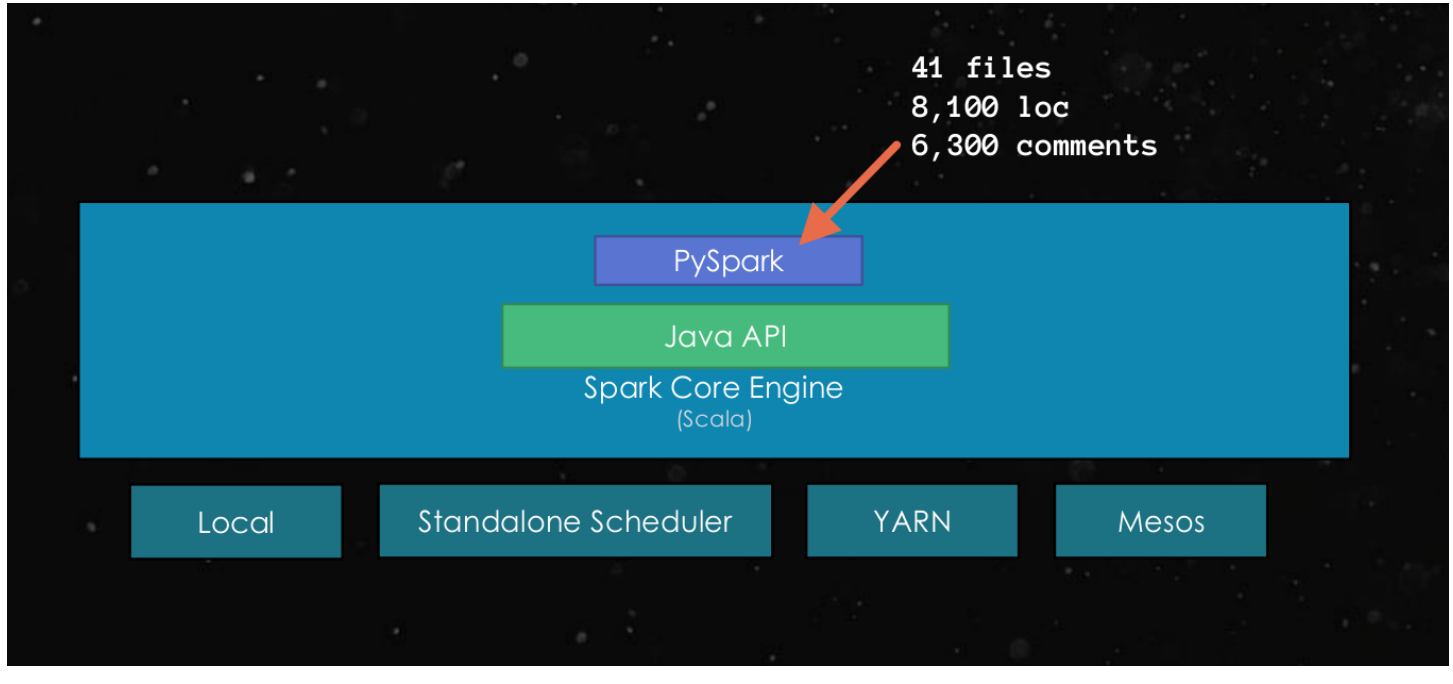
Spark Universe



PySpark

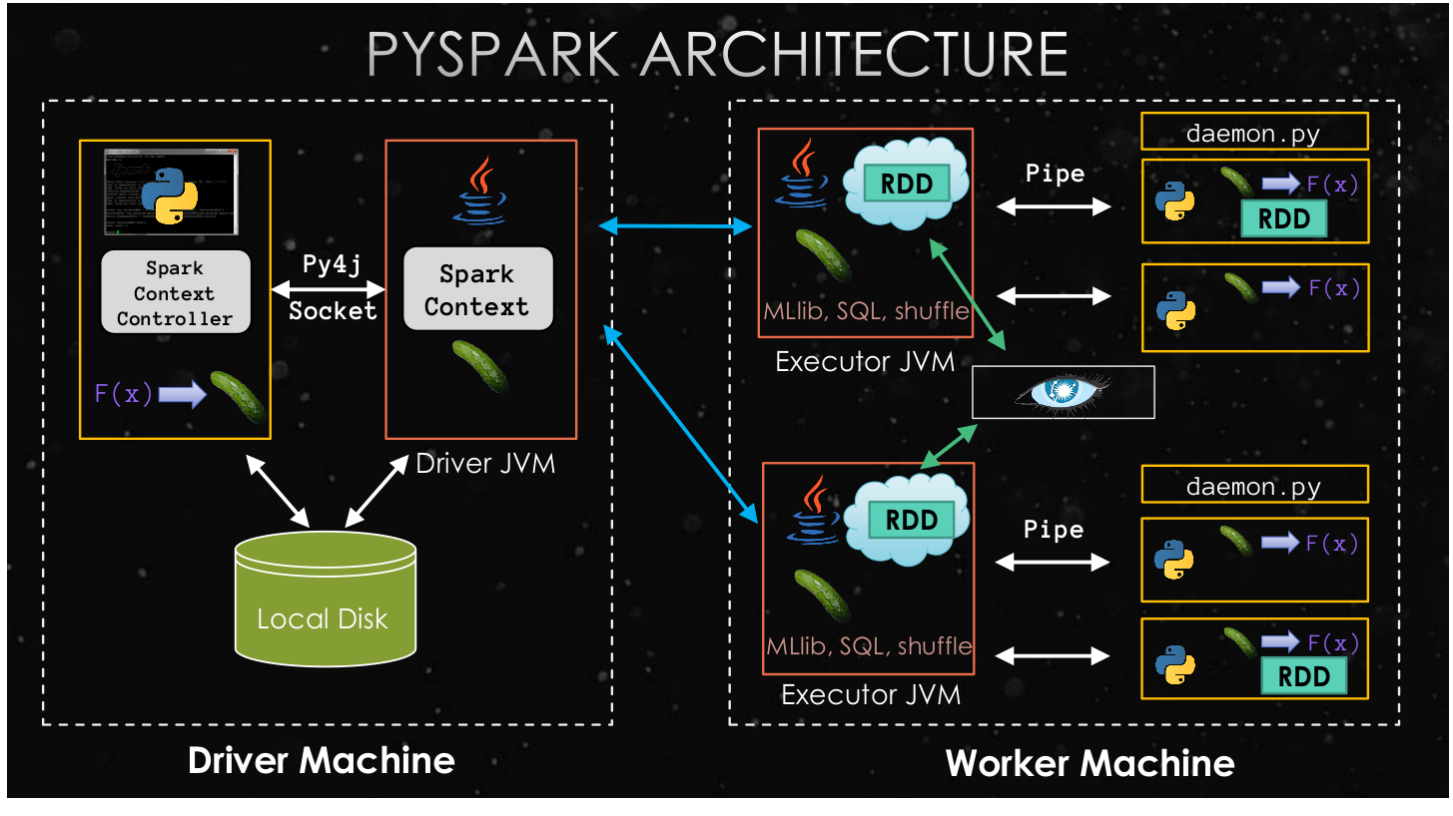
PySpark Stack

- PySpark sits on top of the Java API
- Call Java methods from Python code (as if they where Python methods)



PySpark Architecture

- Pipe on worker machine: custom pipe for high throughput
- Do not read data in Python: would be high I/O so executor JVM on Worker machine reads directly
- Py4j is slow for high throughput: use local disk to ship datasets at driver machine



Python Implementation

- CPython (default)
- pypy (use if you don't need C libraries, if you just use plain Python code)
  - JIT: 20-3000% faster
  - Less memory
  - CFFI (C Foreign Function Interface) support

Spark Competitors (extract)

- Apache MapReduce
- Apache Flink: Stream and batch data processing
- **SQL**
  - Apache Hive
  - Apache Pig
- **Streaming**
  - Apache Storm
  - Apache Apex
  - Apache Samza
- **Machine Learning**
  - Apache Mahout
- **Graph**
  - Apache Giraph